Exhibit B

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, vs. GOOGLE INC. Defendants.	Case No. 3:10-cv-03561-WHA

REPLY EXPERT REPORT OF JOHN C. MITCHELL REGARDING PATENT INFRINGEMENT

SUBMITTED ON BEHALF OF PLAINTIFF
ORACLE AMERICA, INC.

VII. THE ’520 PATENT

A. Rebuttal of Non-Infringement Arguments

89. Prof. Parr states in paragraph 57 that “software cannot infringe a method claim.” Prof. Parr appears to be making a legal argument, not a technical or factual argument based on his expertise. In any event, the Android dx tool carries out the claimed method steps when used as Google intends (*e.g.*, a program developer using the Android dx tool to build Android applications or system images). Because Google provides the dx tool in binary form (the file dx.jar) to developers who download the Android SDK, it can be inferred that the dx program flow is not altered from that expressed in Google’s source code when Android developers use the dx tool to create .dex files.

90. The thrust of Prof. Parr’s report is the assertion that the Android dx tool does not “simulat[e] execution of the byte codes . . . to identify the static initialization of the array,” because the “parseNewarray method in the BytecodeArray.java file . . . performs a pattern matching function to identify the static initialization of the array.” (Parr ’520 Report ¶ 63.) Prof. Parr further states that “When the code in the Simulator class sees a ‘newarray’ bytecode, the Simulator class ***transfers control*** to the parseNewarray method,” and further that parseNewarray “does not use any simulation of execution to extract these initialization values.” (*Id.* at ¶¶ 73-76.) Prof. Parr further argues that the actions by parseNewarray are “pattern matching” and not “simulat[ing] execution.” *Id.* at ¶ 82.)

91. I disagree with Prof. Parr and find, as described in my initial report, that the Android dx tool does indeed “simulat[e] execution of the byte codes . . . to identify the static initialization of the array,” as recited by claim 1 (and similarly claims 12 (play executing) and 20 (simulating execution)). Android’s Simulator.java, which is called “simulator,” simulates execution of the bytecode to statically initialize the array. The method processes the instructions one at a time and uses operands given by the instructions to identify the initializations without execution of the bytecodes. For instance, as Prof. Parr illustrates in his trace of Simulator.java,

shown in paragraph 66 of his report, the dx tool proceeds through the array instructions (*e.g.*, 5,4,3,2,1), one at a time, recording values associated therewith, thereby using the process to identify a static initialization of the array without actually executing the bytecodes as recited by the claims.

92. parseNewarray is part of the functionality and operation of Simulator.java, the "class which knows how to simulate the effects of executing bytecode," and which Prof. Parr concedes simulates bytecodes. It is not uncommon that functionality be carried out or span multiple classes of files. The dx tool was designed to receive class files that had been created by a Java compiler, and parseNewarray interprets the array initialization bytecodes that result from compilation to determine the static values, which are then used to create (and store) one or more instructions requesting the static initialization of the array. The claims do not require the simulating step to succeed in identification of the static initialization of the array in all cases, which is why Prof. Parr's example in his Appendix A, Section II is not evidence of noninfringement. Actually, it is evidence of infringement, because the operation of the Simulator on his modified T.class file produced instructions requesting the static initialization of the array, which were stored in the output .dex file. I note that those instructions were fewer than the original number of bytecodes from the .class file, which is the purpose of the '520 patent.

B. Mr. Poore's Performance Analysis

93. Prof. Parr criticizes the experiments of Prof. Poore, primarily attacking whether or not the tested programs are real world examples, in paragraphs 121-134. Prof. Parr appears to have singled-out applications that have small statically initialized arrays. Even in these applications, most do have static arrays that benefit from the use of the claimed invention of the '520 patent. I note also that it is not just applications that benefit from the '520 patent, but the Android system code itself. As Dan Bornstein explained at Google I/O in 2008, the use of the dx tool brings Android substantial benefits:

> So, sometimes you really need to have just a big array of data and if you've ever looked at what something like this looks like at a, at a, in

> a .class file, it's not pretty. ... And each time you add another element to say an int, an int array, it's 11 bytes of code and constant combined and another, another four instruction dispatches. … So knowing that, this is what we do in a dex file. This is the same code, turned, which has been translated into a .dex file. You can see that for, in this case we're really, we're only using 46 bytes for, for this example and as you add elements to that int array it's just four more bytes per element which is exactly the data represented. And we only have to interpret one opcode to do that entire initialization and that's the third one down, the fill array data. … … And this is both a speed and a, a space efficiency win. ***Measured on our system libraries it saves us something like a 100K***.

Google I/O 2008 Video entitled "Dalvik Virtual Machine Internals," presented by Dan Bornstein, *available at* http://developer.android.com/videos/index.html#v=ptjedOZEXPM (emphasis added).

IX. THE ’447 AND ’476 PATENTS

A. Reply to Mazières ’447 and ’476 Report

1. Dalvik Provides Security Using java.security Package

108. Prof. Mazières advances the position that Java security should not be considered a security mechanism for Android because it does not provide security when portions of Android applications are implemented using native code. (Mazières ’447 and ’476 Rebuttal Report ¶¶ 20, 24.) I believe Prof. Mazières is referring to portions of applications developed using the Android NDK because this is the recommended way to develop native portions of applications, and he does not identify any other way to introduce native code into an application.

109. First, of the five asserted claims in the ’447 and ’476 patents, only one claim recites that the claimed invention is “for providing security.” (*See* ’476, claim 14.) Therefore, most claims do not explicitly require security to be provided, and even then, I understand that claim preambles are generally not limiting. Even for the single claim that does mention “providing security” in its preamble, however, it does not require that the invention be thoroughly secure and block all avenues of attack. Security at any level is acceptable. In addition, a variety of conventional security mechanisms, such as password-based authentication, are vulnerable to some attacks.

110. Defense in depth is an accepted principle of secure system design that involves combining security mechanisms. Android development may involve Java code developed using the SDK and additional native code developed using the NDK. It makes sense to combine a Java security mechanism for Java portions of an application with another security mechanism for native code. In addition, the principle of least privilege can be enforced for Java code, isolating components of the system, using the ’447 and ’476 patented inventions, regardless of the presence of native code.

111. Prof. Mazières suggests that alternative security models exist. (Mazières ’447 and ’476 Rebuttal Report ¶¶ 62-64.) However, Prof. Mazières points to alternatives that are still

under research and not yet fully developed or implemented on the same scale as the Java security provided by the accused portions of Android.

B. The Claims Do Not Require That There Be No Additional Programming

112. Prof. Mazières has been informed that "in order to prove infringement of a computer readable medium claim of a patent, Oracle must demonstrate that the instructions carried by the accused computer-readable media can perform each of the claimed steps without additional programming." (*Id.* ¶ 25.) I do not understand that to be correct. Prof. Mazières points to the fact that I have not taken steps to verify whether the claimed steps can be performed to support his assertion that the code I identify cannot be used without additional programming. However, the claim does not require that the code be used without additional programming.

113. For example, claim 10 of the '447 patent claims "[a] computer-readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps of…." ('447, 14:35-39.) The other asserted claims of the '447 and '476 patents are similar. According to the claim language, the instructions need not be used or even executed. Rather, the claims require only that the instructions be carried on the computer-readable medium. The correct test for infringement of the claims as drafted is not whether the code is instantiated or executed. Rather, the correct test is simply whether the code is available on computer-readable media. As shown by my infringement analysis, Android code that is capable of performing the required functionality is stored on computer-readable media.

114. Further, Dalvik preloads the classes in the java.security package that result in infringement. Given all the evidence on Google's goals of saving space and creating a system that is as small and fast as possible, why did Google include the infringing code, taking up valuable space on the device? Why is the infringing code loaded, taking up valuable startup time? Even without additional programming to "turn on" the functionality of the Java security

framework provided on Android devices, infringing code is "carried" on computer-readable media.

115. Even if the SecurityManager is disabled, the Java security framework may still be used. For example, "the static methods in AccessController are always available to be called." (L. Gong et al., Inside Java 2 Platform Security 109 (2nd Ed. 2003).) Even if no SecurityManager is instantiated, Android code includes AccessController.java and the static methods associated with the class may be called. Therefore, the functionality of the Java security framework is accessible via the AccessController.

C. The Claims Are Written In An "Open" Manner

116. Prof. Mazières states that "claim 10 recites the practice of steps in a closed rather than open manner, i.e., without the benefit of inclusive phrasing such as 'comprising' in connection with the recitation of the steps." (Mazières '447 and '476 Rebuttal Report ¶ 81.) I disagree with this statement. Each of the claims of the '447 and '476 patents recites the phrase "including instructions." This phrase implies that the claims are written in an open manner, not a closed manner. Other instructions may also be present (i.e., "included") in the accused instrumentality and the instrumentality will still infringe.

D. The Compatibility Test Suite Highlights That Android Devices Contain Infringing Code

117. In paragraph 20, Prof. Mazières points to the evidence that developers execute the Compatibility Test Suite (CTS). Prof. Mazières states that "the CTS code is not installed on Android devices and thus, is not used to provide security." (*Id.* ¶ 20.) While this statement is true, Prof. Mazières misunderstands the purpose of my discussion about the CTS. Google requires Android devices to pass the CTS before they may be branded with the Android name, among other things the device makers receive from Google. Prof. Mazières admits that the CTS is used "with the purpose of verifying the capabilities of a set of software to be installed on an Android device." (*Id.*) One set of capabilities the CTS verifies are the capabilities associated with the security framework discussed in my opening infringement report. Therefore, the fact

X. SUMMARY OF OPINIONS AND CONCLUSION

123. It is my opinion that Google's experts have failed to demonstrate that any asserted claim of the patents-in-suit is not infringed.

124. It is also my opinion that Google's experts have failed to demonstrate that Google is not liable for direct and indirect infringement in the manner described in my Opening Patent Infringement Report.

125. It is further my opinion that Google's experts have failed to demonstrate that the patents-in-suit do not form the basis for consumer demand for Android by developers and end-users.

126. It is also my opinion that Google's experts have failed to demonstrate that the patents-in-suit were not necessary to Android achieving satisfactory performance and security, once Google decided to adopt the Java execution model in Android.

Dated: September 1, 2011

[illegible]

John C. Mitchell